Padded Frames: A Novel Algorithm for Stable Scheduling in Load-Balanced Switches

Juan José Jaramillo
Coordinated Science Laboratory
Department of Electrical
and Computer Engineering
University of Illinois
at Urbana-Champaign
Email: jjjarami@uiuc.edu

Fabio Milan
Dipartimento di Elettronica
Politecnico di Torino
Turin, Italy
Email: milan@mail.tlc.polito.it

R. Srikant
Coordinated Science Laboratory
Department of Electrical
and Computer Engineering
University of Illinois
at Urbana-Champaign
Email: rsrikant@uiuc.edu

***Abstract*— The load-balanced Birkhoff–von Neumann switching architecture consists of two stages: a load balancer and a deterministic input-queued crossbar switch. The advantages of this architecture are its simplicity and scalability, while its main drawback is the possible out-of-sequence reception of packets belonging to the same flow. Several solutions have been proposed to overcome this problem; among the most promising are the Uniform Frame Spreading (UFS) and the Full Ordered Frames First (FOFF) algorithms. In this paper, we present a new algorithm called Padded Frames (PF), which eliminates the packet reordering problem, achieves 100% throughput, and improves the delay performance of previously known algorithms.**

I. Introduction

The continued explosive growth of the Internet fuels research in the problem of designing scalable, fast switching architectures. A simple and scalable solution is the load-balanced Birkhoff–von Neumann algorithm (see Fig. 1), first introduced by Chang et al. in [1], [2], which is implemented using a two-stage architecture. The first stage is a load balancer, which spreads the input traffic uniformly among the intermediate VOQs. The second stage is a deterministic input-queued crossbar switch. Both stages execute the same sequence of N configurations (where N is the size of the switch, i.e., the number of input and output ports), such that every input and every intermediate queue is served deterministically $1/N$-th of the time. It has been proved that this switch can provide 100% throughput under a broad class of traffic patterns. The main drawback is that, due to the existence of N parallel paths from each input i to each output j, it is possible for packets belonging to the same flow (i, j) to get out of order. To avoid having to reorder the packets at the output, several modifications to the basic load-balanced architecture have been proposed: First Come First Served (FCFS) [2], Earliest Deadline First (EDF) [2], [3], Full Frames First (FFF) [4], Application Flow-Based Routing (AFBR) [5], Uniform Frame Spreading (UFS) [5], and Full Ordered Frames First (FOFF) [5], [6].

The first author's work was supported by a Fulbright fellowship and the second author's participation in the project occurred during a visit to the University of Illinois.

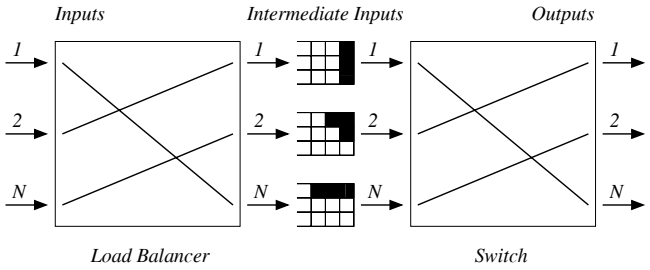

Fig. 1. The basic load-balanced architecture

The rest of the paper is organized as follows. Section II describes the Padded Frames (PF) algorithm. In Section II-A we prove that our algorithm is stable under *any* admissible traffic pattern. In Section III we propose a modification to the original PF algorithm which enhances its performance, and in Section III-A we prove that it is stable. Section IV presents some simulations that confirm the analysis, and shows the performance improvement of Padded Frames with respect to existing algorithms. Section V contains some comments about the implementation complexity of PF. Finally, Section VI provides concluding remarks.

II. The Padded Frames Algorithm

Padded Frames (PF) is a frame-based scheduling algorithm for load-balanced switching architectures. We use the term *frame* to refer to either a set of N packets, or to a set of N time-slots, where N is the size of the switch, i.e., the number of input and output ports in the switch. Every input of the first stage is equipped with N VOQs. An arriving packet to input i directed to output j is placed in VOQ_{ij}. PF prevents packet reordering by assuring that all the intermediate input queues have the same length. In order to do so, every N time-slots PF schedules for service with higher priority those input queues which have a full frame, choosing among them in round-robin order. If there is always a full frame to be transferred at each

input port, PF behaves exactly as the UFS algorithm. When no full frames can be found at an input port, then PF chooses the largest nonempty queue from the VOQs at the input. As in the FOFF algorithm, serving the VOQs which do not have a complete frame, albeit at a lower priority, avoids long delays if the load is low, or starvation of smaller flows.

The distinctive feature of our algorithm is that, in order to deliver packets in sequence to the outputs, if it is necessary to transfer an incomplete frame of length $0 < F < N$; then PF will insert $N - F$ fake packets to obtain a *padded frame*. Sending such fake packets could eventually lead to an increase in the arrival rate to the intermediate inputs, and therefore produce instability. One of the contributions of this paper is to show that if the amount of padding is controlled, then this does not result in either instability or throughput loss.

The intuitive reason for the stability of PF is that padding is necessary only at light loads, so it does not result in a throughput reduction. Conversely, at heavy load, the probability of having a full frame is high, hence padding is not needed nor desired.

From an architectural point of view, since PF prevents packet reordering, it does not require an output buffer to resequence the packets.

As in the load-balanced Birkhoff–von Neumann switch, in PF each of the two switching stages goes through the same predetermined cyclic shift configuration. The chosen configuration in our algorithm will assure that every input will send packets starting from intermediate input 1 all the way through intermediate input N in an increasing order, and that every output will serve intermediate inputs in the same order.

Scheduling proceeds according to the following steps:

1) An arriving packet to input i directed to output j is placed in input VOQ_{ij}.
2) Each of the two switching stages of the load-balanced switch goes through the following predetermined cyclic shift configuration: at time t, input i is connected to intermediate input $[(i+t-1) \mod N]+1$ and intermediate input $[(j+t-2) \mod N]+1$ is connected to output j.
3) At every input, and every N time slots, the algorithm will search in round-robin order among the input VOQs that have at least N packets (one full frame).
4) If there exists no such input VOQ, the algorithm searches among the nonempty queues looking for the largest one. Without loss of generality, consider only input i. Assume that input VOQ_{ik} is the largest queue. Let L_k be the length of the k-th VOQ of intermediate input 1, and let T be an integer parameter. Then perform the following test. If $L_k < T$, then schedule input VOQ_{ik} to be served, and go to step (5). Else, do not serve input VOQ_{ik}.
5) When spreading packets from an input to the intermediate inputs, always start sending packets to intermediate input 1, and then continue serving intermediate inputs in an increasing order until intermediate input N is reached. When serving an input VOQ that was scheduled in step 4, if there is no packet to send, send a fake packet to the intermediate inputs. Otherwise, send the stored packets.
6) In the second stage, PF uses N VOQs per intermediate input (one per each output), and behaves exactly like the basic load-balanced switch: when intermediate input m is connected to output n, intermediate input VOQ_{mn} will be served.

Before concluding this section, we make a few comments on the PF algorithm. First, notice that at step 4, through the parameter T, we regulate the amount of padding present in the intermediate inputs. The idea is that the lengths of the queues is a form of congestion indication. Indeed, if the intermediate queues are long, then transferring fake packets could result in a throughput loss. On the other hand, if the intermediate queues are short, then the fake packets do not adversely impact switch stability. Second, notice that if we set $T = 0$, then we are back to the UFS algorithm. Thus, PF is a generalization of UFS. Third, notice that when we are performing the test on the queue length to limit the number of padded frames in the intermediate input, we only have to look at the number of packets in intermediate input 1. Since packets are uniformly spread on all the N intermediate inputs, and either 0 or N (real or fake) packets are transferred, all the intermediate inputs have the same length. This information can be very useful in reducing the amount of information exchanged by the line cards. Finally, a comment on step 5. Suppose that a packet P directed to output j arrives at input i, exactly when the input i is serving the VOQ j. Suppose that the length of the input VOQ_{ij} is strictly less than N, so that padding is necessary. Then, in this case, packet P can be used to preempt the fake packet which was scheduled to be transfered. Generally, this means that fake packets could be intertwined among real packets, thus improving the efficiency of the switch.

A. Stability

As is the case in AFBR, FFF, UFS, and FOFF, for simplicity we assume that all incoming packets are segmented into fixed-size cells, which we will simply call packets, and then reassembled at the output before leaving the switch. We also assume that there is at most one arrival per time slot at each input, and arrivals occur first in a time slot before departures.

We will prove that our algorithm achieves 100% throughput by comparing the load-balanced switch using the PF algorithm with a hypothetical switch where the second switching stage of the load-balanced switch is replaced with a work-conserving server.

The following lemma, proved in [7], will be useful in our proof:

Lemma 1: Consider a work-conserving server and let $A(t)$ and $D(t)$ respectively denote its cumulative number of arrivals and services until time t. Assume that its service capacity is one packet per time slot. Then for all $t \geq 0$,

$$D(t) = \min_{0 \leq s \leq t} [A(s) + t - s].$$

Theorem 1: The PF algorithm has the same throughput, i.e., the same average number of packets served per time slot, as an ideal output-queued switch irrespective of the arrival process.

Proof: Without loss of generality, consider only output k and suppose that at time $t = 0$ there are no packets in the queues. Define the following variables:

- $A_k(t)$: cumulative number of packet arrivals up to time t at the inputs of the load-balanced switch destined to output k.
- $B_k(t)$: cumulative number of packet arrivals up to time t at the intermediate inputs of the load-balanced switch destined to output k.
- $C_k(t)$: cumulative number of packet departures up to time t of the load-balanced switch at output k.
- $C_k^{WC}(t)$: cumulative number of packet departures up to time t of a work- conserving server of unit capacity with arrivals given by $B_k(t)$.
- $C_k^{OQ}(t)$: cumulative number of packet departures up to time t at output k of an output-queued switch with arrivals given by $A_k(t)$.
- $q'_k(t)$: queue length at the inputs of the load-balanced switch for packets destined to output k.
- $q''_k(t)$: queue length at the intermediate inputs of the load-balanced switch for packets destined to output k.

Since the cumulative number of departures is always less than or equal to the cumulative number of arrivals, it is easy to see that

$$B_k(t) \geq C_k^{WC}(t) \tag{1}$$

Observe that at any input, from the time instant the last packet in a full frame arrives until the next time the VOQ is considered for scheduling, there can be at most $N-1$ time slots. It is also important to notice that at every time slot, only one input can send a packet to the first intermediate input, so the maximum delay a VOQ can experience from the instant it is scheduled to be served until it actually gets service is $N-1$ time slots.

Note that there can be at most $N(N-1)$ packets in any input without having a full frame. Also, the presence of a full frame ensures that the input will start serving packets (with a delay bounded by $N-1+N-1 = 2N-2$, as explained in the previous paragraph); thus, at any time instant the maximum number of packets in any input is $N(N-1)+2N-2 = N^2+N-2$ (after service) and this number cannot increase any further. Hence, all inputs have at most N^3+N^2-2N packets destined to output k. Thus,

$$q'_k(t) = A_k(t) - B_k(t) \leq N^3 + N^2 - 2N \tag{2}$$

Now, let us analyze the second switching stage. Since we always uniformly send packets to the intermediate inputs, by the pigeonhole principle if the intermediate input queue size is greater than or equal to TN one can only schedule full frames. Thus, if there are TN or more packets destined to output k at the intermediate inputs, then the PF algorithm can only schedule real packets to the intermediate stages. Further, if the number of packets destined to output k is less than or equal to $TN-1$, then each input can send N packets (whether fake or real) over the next N time slots to the intermediate stage. This means that when

$$q''_k(t) \geq N^2 + TN$$

the load-balanced switch will only send full frames to the intermediate inputs.

At any time t the load-balanced switch can only be in either of the following two states:

$$q''_k(t) < N^2 + TN \text{ or} \tag{3}$$

$$q''_k(t) \geq N^2 + TN \tag{4}$$

For a time $\hat{t}$ such that (3) holds we have the following:

$$C_k(\hat{t}) \overset{(3)}{>} B_k(\hat{t}) - N^2 - TN$$

Hence:

$$\begin{aligned} C_k(\hat{t}) &\overset{(1)}{\geq} C_k^{WC}(\hat{t}) - N^2 - TN \\ &\overset{Lemma\ 1}{=} \min_{0\leq s\leq \hat{t}} \left[B_k(s) + \hat{t} - s\right] - N^2 - TN \\ &\overset{(2)}{\geq} \min_{0\leq s\leq \hat{t}} \left[A_k(s) + \hat{t} - s\right] - N^3 - 2N^2 - (T-2)N \end{aligned} \tag{5}$$

And since an output-queued switch can be modeled as a work-conserving server with unit capacity we have

$$= C_k^{OQ}(\hat{t}) - N^3 - 2N^2 - (T-2)N$$

Thus for (3) we have

$$C_k(\hat{t}) \geq C_k^{OQ}(\hat{t}) - N^3 - 2N^2 - (T-2)N \tag{6}$$

The only way to get to state (4) is through (3), where (5) holds for the first time slot when the new state is reached. After that, the switch cannot generate fake packets any longer; it only serves the remaining ones on the queues, which can be at most $(T+N)(N-1)$. Thus the difference in service between our algorithm and the work-conserving server can only increase up to $(T+N)(N-1)$ packets. Therefore, for any time $\tilde{t}$ such that (4) holds, we have

$$\begin{aligned} C_k(\tilde{t}) &\geq C_k^{WC}(\tilde{t}) - 2N^2 - T(2N-1) + N \\ &\overset{Lemma\ 1}{=} \min_{0\leq s\leq \tilde{t}} \left[B_k(s) + \tilde{t} - s\right] - 2N^2 - T(2N-1) + N \\ &\overset{(2)}{\geq} \min_{0\leq s\leq \tilde{t}} \left[A_k(s) + \tilde{t} - s\right] - N^3 - 3N^2 - T(2N-1) + 3N \\ &= C_k^{OQ}(\tilde{t}) - N^3 - 3N^2 - T(2N-1) + 3N \end{aligned}$$

Thus:

$$C_k(\tilde{t}) \geq C_k^{OQ}(\tilde{t}) - N^3 - 3N^2 - T(2N-1) + 3N \tag{7}$$

From (6) and (7), since the number of packets served by PF is within a constant of the number of packets served by an ideal output-queued switch, it follows that PF has the same throughput of the output-queued switch, irrespective of the arrival process.

■

III. Improvements to PF

Although the original algorithm is stable as proved in Section II-A, it can be further improved. Remember that when fake packets are being sent, they have to be processed at the intermediate inputs as real packets, and discarded only when they arrive at the outputs. The time slots consumed serving fake packets lead to an increase in the average delay of the switch, so it is beneficial to further control the maximum amount of fake packets that are generated.

A first idea is to keep track of the number of fake packets in the intermediate inputs, and when they reach a threshold the algorithm is only allowed to send full frames. The problem with this approach is that the counter which keeps track of fake packets has to be updated up to $2N$ times every time slot, once when a fake packet is being sent to the intermediate inputs and once when the packet is discarded at the outputs. Further, this approach is not convenient under high load, when it is extremely important to minimize any additional processing due to fake packets.

This concern lead us to consider a different approach: keeping track of the number of padded frames at the intermediate inputs destined for every output. Now there will be N counters, one per output, and they will only need to be updated every N time slots, either during scheduling or at the end of serving a padded frame at the output. Additionally, since the number of padded frames, rather than fake packets, is being upper-bounded, under heavy load the number of fake packets in the intermediate inputs is reduced, because the probability of serving a padded frame with a large number of fake packets decreases.

This intuition leads us to the following modifications to the original algorithm. The improved algorithm will now be called PF+.

- Modify step (4) as follows. Assume that at input i input VOQ_{ik} was scheduled to be served. Let PF_k be a counter that keeps track of the number of padded frames in the intermediate inputs destined to output k. Let W be an integer parameter, and let L_k and T be defined as in the original PF algorithm. Then check if $PF_k < W$ *and* if $L_k < T$. If so, then keep the schedule, and update $PF_k \leftarrow PF_k + 1$. Else, do not schedule input VOQ_{ik}.
- Modify step (5) as follows. When serving an input VOQ that was scheduled in step 4, if there is no packet to send, send a fake packet to the intermediate inputs. Otherwise, send the stored packets. During the service time if packets arrive to the input VOQ in such a way that it is not necessary to send fake packets, update counter $PF_k \leftarrow PF_k - 1$.
- Add a new step after (6): At output k, once it completely receives a padded frame from the intermediate inputs, update $PF_k \leftarrow PF_k - 1$.

Notice that if either $W = 0$ or $T = 0$, then the PF+ algorithm reduces to to the UFS algorithm, since no padding is allowed. Hence, PF+ is a generalization of UFS.

A. Stability of PF+

In PF+ we are adding one more constraint to send padded frames, so the number of padded frames sent can only decrease with respect to PF. With that in mind, we state the following theorem:

Theorem 2: The PF+ algorithm has the same throughput (i.e., the same average number of packets served per time slot) as an ideal output-queued switch irrespective of the arrival process.

Proof: The proof follows from the proof for Theorem 1. In this case the maximum amount of fake packets in the intermediate inputs can be at most

$$K = \min\{(T+N)(N-1), W(N-1)\}.$$

For a time $\hat{t}$ such that the switch is in state (3), the result still is the same as in theorem 1:

$$C_k(\hat{t}) \geq C_k^{OQ}(\hat{t}) - N^3 - 2N^2 - (T-2)N \qquad (8)$$

And for a time $\tilde{t}$ such that the switch is in state (4) we have the following:

$$\begin{aligned} C_k(\tilde{t}) &\geq C_k^{WC}(\tilde{t}) - N^2 - TN - K \\ &\overset{Lemma\ 1}{=} \min_{0\leq s\leq \tilde{t}} \left[B_k(s) + \tilde{t} - s\right] - N^2 - TN - K \\ &\overset{(2)}{\geq} \min_{0\leq s\leq \tilde{t}} \left[A_k(s) + \tilde{t} - s\right] - N^3 - 2N^2 - (T-2)N - K \\ &= C_k^{OQ}(\tilde{t}) - N^3 - 2N^2 - (T-2)N - K \end{aligned}$$

Thus:

$$C_k(\tilde{t}) \geq C_k^{OQ}(\tilde{t}) - N^3 - 2N^2 - (T-2)N - K \qquad (9)$$

From (8) and (9), since the number of packets served by PF+ is within a constant of the number of packets served by an ideal output-queued switch, it follows that PF+ has the same throughput as that of the output-queued switch, irrespective of the arrival process.

■

IV. Simulations

In all our simulations, we use a Bernoulli traffic model, where the matrix of arrival rates at input i for output j $\{\lambda_{ij}\}$ has random entries subject to

$$\sum_j \lambda_{ij} = \lambda \ \forall i \text{ and } \sum_i \lambda_{ij} = \lambda \ \forall j$$

The parameter λ indicates the load on the switch and will be varied to study the performance of the switch at various loads. The unit of time for all simulations is one time slot.

Since UFS and FOFF seem to be the most promising algorithms to avoid resequencing in load-balanced switches, we compared PF and PF+ to these two algorithms.

In Fig. 2 we compare both PF and PF+ against UFS. As can be seen, the improvement under low load is quite significant. This is because UFS can only send packets when there is a full frame, while in our case we can send padded frames. In Fig. 3

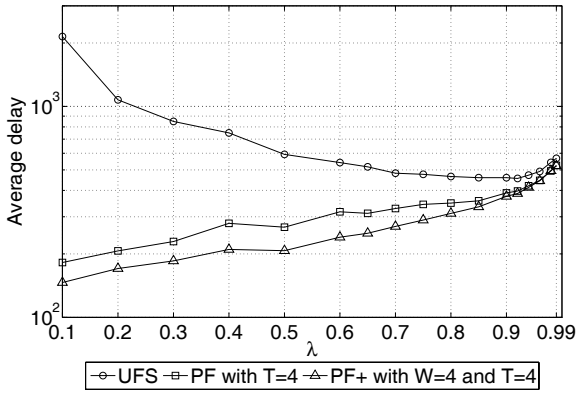

Fig. 2. Average delay comparison against UFS for a 50×50 switch under nonuniform independent Bernoulli arrivals

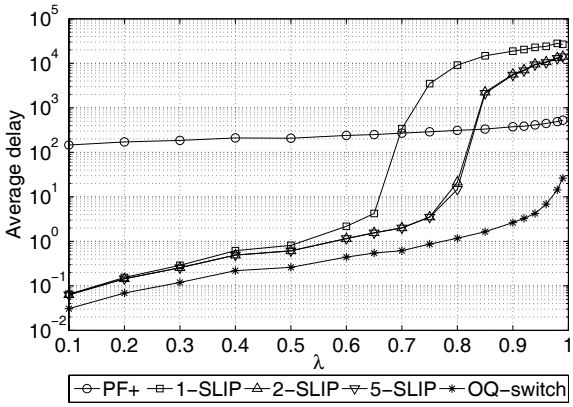

Fig. 4. Average delay comparison against iSLIP for a 50×50 switch under nonuniform independent Bernoulli arrivals

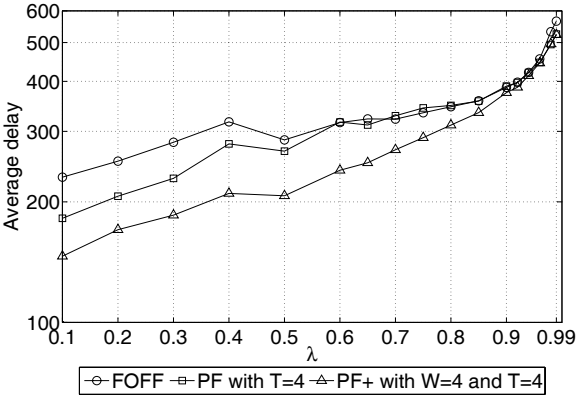

Fig. 3. Average delay comparison against FOFF for a 50×50 switch under nonuniform independent Bernoulli arrivals

we compare our algorithms with FOFF. Still our algorithms show a significantly better performance.

While programming the FOFF algorithm for our simulations we found out that if a crossbar model is used for the two switching stages, as opposed to the mesh model proposed in [5], one can no longer use FIFO queues for the input buffering in the FOFF. In fact, we noticed that when using a FIFO queue, the throughput of the algorithm seems to be limited to around $2/3$ of the maximum possible throughput of the switch. The intuition behind this is that since FOFF keeps track of the last intermediate input the flow from input i to output j sent a packet, it can only start sending packets once the crossbar is in the right configuration. Due to this, it is possible to come up with an adversarial traffic pattern that will make the algorithm always wait for up to $N - 1$ time slots before it can start sending packets for the VOQ that is in service, even if you are sending full frames only. This would make the queues at the inputs to grow unbounded.

As suggested to us by one of the developers of the FOFF algorithm (Isaac Keslassy), this problem can be solved by using Random Access Memory (RAM) based queues both at the input and output. It is important to note that our simulations were not run using the mesh model for the two switching stages of the load-balanced switch, since we only focused on the crossbar model.

Finally, we compare the load-balanced switch with PF+ against another architecture for the input-queued switch and the ideal output-queued switch. For the input-queued switch we decided to compare our algorithm to iSLIP [8], which is known to be easy to implement and has been successfully deployed commercially. In [8] it is proven that iSLIP converges in at most N iterations, where N is the switch size, but it is stated that through simulations they have found that usually the expected number of iterations needed to converge is less than or equal to $\log_2 N$. For our simulations, we compare our algorithm against 1-SLIP, 2-SLIP, and 5-SLIP.

In Fig. 4, where we used nonuniform independent Bernoulli arrivals, it can be observed that although in the low load region PF+ performs poorly compared to both iSLIP and the output-queued switch, in the high load region our algorithm outperforms iSLIP, showing that our algorithm tends to be more stable under high loads.

In [8] it is stated that a more realistic traffic model is one with bursty arrivals. This is due to the fact that real network traffic is highly correlated and packets tend to arrive in bursts, corresponding to packetized files. So we decided to compare the two algorithms, using the same bursty traffic model used in [8]. Packets are generated by an on-off arrival process modulated by a two-state Markov chain such that at every input the source generates a busy period of packets (with the same destination) followed by a period of inactivity. The number of packets in the busy and idle periods are geometrically distributed, and the destination for a burst is uniformly distributed over all outputs.

In Fig. 5 we present the result of the simulations, where the mean of the busy periods is 128 packets. Although not shown here, we also run simulations with busy period means of 16, 32, and 64 packets. We were able to observe that the bigger the bursts the closer PF+ approaches the output-queued average delay, especially at higher loads. It should be noted that in high loads PF+ still outperforms iSLIP, although not as

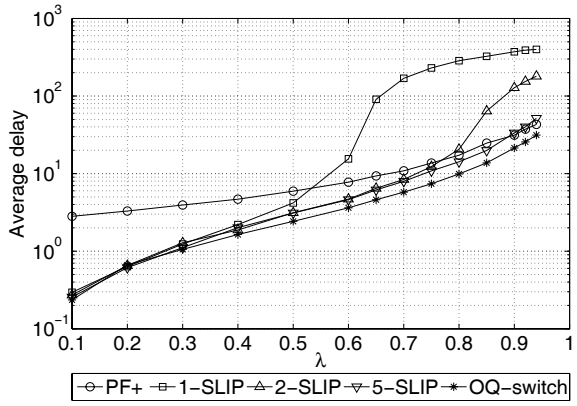

Fig. 5. Average delay comparison against iSLIP for a 50×50 switch under bursty arrivals

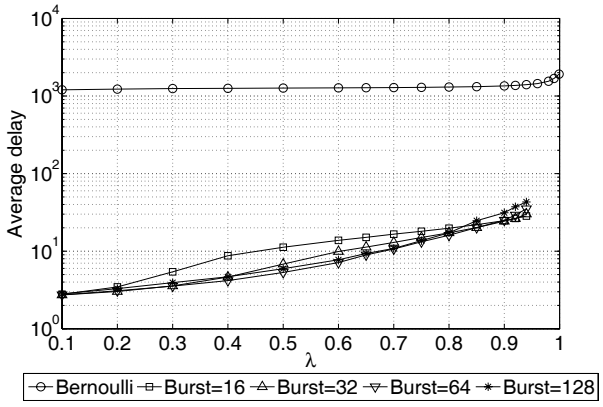

Fig. 6. Variation of average delay of PF+ for a 50×50 switch under different traffic models

dramatically as in the case of Bernoulli arrivals. We point out that PF+ is provably stable at all loads whereas such a result has not been established for iSLIP. We also run simulations of both UFS and FOFF under this traffic model and our algorithm continued to have a better average delay than these two algorithms. These results are not shown here since the figures are quite repetitive. It is interesting to note that our algorithm has a performance similar to that of the output-queued switch for bursty arrivals. This is due to the fact that under bursty traffic the algorithm does not generate as many fake packets as under Bernoulli traffic, so the performance penalty incurred decreases substantially, as can be seen in Fig. 6.

V. Implementation Complexity

We summarize the implementation characteristics of PF and PF+ below:

i. Both are easy to implement in either a crossbar or mesh architecture, require only FIFO queues, and do not need reordering buffers at the output ports.

ii. One may argue that a possible disadvantage is that one has to choose the parameters T and W to ensure good performance. For example, $T = W = 0$ reduces PF+ to UFS, which has poor performance at low loads. On the other hand, we have shown that T and W are quite insensitive to the switch size N and the traffic model. Thus, having to choose these parameters does not seem to be a significant issue.

VI. Conclusions

In this paper we have presented Padded Frames, a novel scheduling algorithm for load-balanced Birkhoff–von Neumann switching architectures. Through the insertion of fake packets into the intermediate input queues, PF prevents packet reordering and starvation of lightly loaded queues. We also presented PF+, a modification of the original algorithm that improves the average packet delay while maintaining the stability and throughput guarantees of PF.

The algorithm has a computational complexity of $O(1)$, requires no speedup, and is decentralized. In addition, the only information that needs to be shared between line cards is the value of N counters (for PF+) and whether the queue lengths of the VOQs at intermediate input 1 have reached the threshold T (for both PF and PF+).

We have shown, by means of both analysis and simulation, that both PF and PF+ are stable for *any* admissible traffic patterns, and their performance is significantly better than those of existing algorithms such as Uniform Frame Spreading and Full Ordered Frames First.

We also compared our algorithm against the input-queued switch implemented with iSLIP and the ideal output-queued switch. From our results we showed that our algorithm outperforms iSLIP under heavy traffic loads, and that in the case of a traffic model with bursty arrivals our algorithm approaches the average delay performance of the output-queued switch.

References

[1] C.S. Chang, D.S. Lee, and Y.S. Jou, "Load balanced Birkhoff–von Neumann switches, part I: One-stage buffering," *Computer Communications*, vol. 25, no. 6, pp. 611–622, 2002.

[2] C.S. Chang, D.S. Lee and C.M. Lien, "Load balanced Birkhoff–von Neumann switches, part II: Multi-stage buffering," *Computer Communications*, vol. 25, no. 6, pp. 623–634, 2002.

[3] C.S. Chang, D.S. Lee, and C.Y. Yue, "Providing guaranteed rate services in the load balanced Birkhoff-von Neumann switches," in *Proc. of IEEE Infocom 2003*, San Francisco, CA, USA, 2003, pp. 1622–1632.

[4] I. Keslassy and N. McKewon, "Maintaining packet order in two-stage switches," in *Proc. of IEEE Infocom 2002*, New York, NY, USA, June 2002, pp. 1032–1041.

[5] I. Keslassy, "The load-balanced router," Ph.D. dissertation, Stanford University, Stanford, CA, USA, 2004.

[6] I. Keslassy, S.-T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown, "Scaling internet routers using optics," in *Proc. of ACM SIGCOMM 2003*, Karlsruhe, Germany, August 2003, pp. 189–200. Also in *Computer Communication Review*, vol. 33, no. 4, pp. 189–200, October 2003.

[7] C.S. Chang, *Performance Guarantees in Communication Networks*. New York: Springer-Verlag, 2000.

[8] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE Transactions on Networking*, vol. 7, no. 2, pp. 188–201, April 1999.